4차시 과제 풀이

문제 목록

문제 번호	문제 이름	출처
BOJ 2630	색종이 만들기	KOI 2001 중등부
BOJ 1074	Z	
BOJ 11819	The Shortest does not...	IZhO 2009
BOJ 4256	트리	2013 ICPC 인터넷 예선
BOJ 10830	행렬 제곱	
BOJ 17297	Messi Gimossi	제3회 천코대 예선
BOJ 1517	버블 소트	
BOJ 1725	히스토그램	
BOJ 2261	가장 가까운 두 점	
BOJ 17613	점프	KOI 2019 1차 고등부
BOJ 1848	동굴 탐험	
BOJ 16901	XOR MST	
BOJ 20503	Haven	제4회 천코대 본선
BOJ 2042	구간 합 구하기	
BOJ 14438	수열과 쿼리 17	
BOJ 2243	사탕상자	
BOJ 5419	북서풍	BAPC 2005
BOJ 2336	굉장한 학생	BalkanOI 2004
BOJ 10167	금광	KOI 2014 중등부
BOJ 16911	그래프와 쿼리	
BOJ 18362	Desert	POI 2014/2015 Stage 2

주의 사항

- BOJ 1517 : 분할 정복 사용 (세그먼트 트리, 펜윅 트리 등등 사용 금지)
- BOJ 1725 : 분할 정복 사용 (스택, 유니온 파인드 등등 사용 금지)
- BOJ 2261 : 분할 정복 사용 (라인 스위핑 사용 금지)

BOJ 2630 색종이 만들기

재귀 함수를 열심히 짜면 되는 문제

만약 현재 함수에서 보고 있는 정사각형이 모두 같은 색이면 정답에 추가하고, 그렇지 않다면 4개로 쪼개서 재귀 호출을 하면 된다.

```
#include <bits/stdc++.h>
using namespace std;

int N, A[222][222], R[2];

void Solve(int r, int c, int sz){
    if(sz == 1){ R[A[r][c]]++; return; }
    int flag[2] = {0};
    for(int i=r; i<r+sz; i++) for(int j=c; j<c+sz; j++) flag[A[i][j]] = 1;
    if(flag[0] && flag[1]){
        Solve(r, c, sz/2);
        Solve(r+sz/2, c, sz/2);
        Solve(r, c+sz/2, sz/2);
        Solve(r+sz/2, c+sz/2, sz/2);
    }
    else if(flag[0]) R[0]++;
    else R[1]++;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) for(int j=1; j<=N; j++) cin >> A[i][j];
    Solve(1, 1, N);
    cout << R[0] << "\n" << R[1];
}
```

BOJ 1074 Z

완전 탐색을 하면 최대 $2^{30} > 10^9$개의 칸을 탐색해야 돼서 시간 초과를 받게 된다.

지금 보고 있는 정사각형을 4개로 쪼갠 뒤, (r, c)가 존재하는 부분으로만 들어가야 한다.

```
#include <bits/stdc++.h>
using namespace std;

int N, R, C;

int Solve(int i, int j, int sz){
    if(sz == 0) return 0;

    int ret = 0;
    int ii = i + (1 << (sz-1));
    int jj = j + (1 << (sz-1));
    if(R < ii && C < jj) return ret + Solve(i, j, sz-1);
    ret += 1 << (sz-1)*2;
    if(R < ii && C >= jj) return ret + Solve(i, jj, sz-1);
    ret += 1 << (sz-1)*2;
    if(R >= ii && C < jj) return ret + Solve(ii, j, sz-1);
    ret += 1 << (sz-1)*2;
```

```
    return ret + Solve(ii, jj, sz-1);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> R >> C;
    cout << Solve(0, 0, N);
}
```

BOJ 11819 The Shortest does not Mean the Simplest

$C \leq 10^{18}$이므로 연산 도중에 `long long` 범위를 벗어날 수 있다. `__int128_t`를 사용하거나 BigInteger를 구현해야 한다.

A^B를 그냥 계산하면 $O(B)$만큼 걸려서 시간 초과를 받게 되고, $A^B = \begin{cases} (A^{B/2})^2 & \text{if } 2 \mid B \\ (A^{(B-1)/2})^2 \cdot A & \text{if } 2 \nmid B \end{cases}$를 이용하면 $O(logB)$만에 계산할 수 있다.

혹은 B를 이진법으로 전개해서 1이 나오는 자리수만 봐도 $O(logB)$에 계산할 수 있다.

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
using LL = __int128_t;

LL Pow(LL A, LL B, LL MOD){
    LL ret = 1;
    while(B){
        if(B & 1) ret = ret * A % MOD;
        B >>= 1; A = A * A % MOD;
    }
    return ret;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    ll A, B, C; cin >> A >> B >> C;
    cout << ll(Pow(A, B, C));
}
```

BOJ 4256 트리

pre-order와 in-order를 알고 있으면 루트 정점과 왼쪽/오른쪽 서브 트리 정점을 각각 알아낼 수 있다.

루트 정점은 pre-order에서 맨 앞에 나오는 정점이다. 왼쪽 서브 트리에 속한 정점은 모두 in-order에서 루트보다 먼저 나오고, 오른쪽 서브 트리에 속한 정점은 모두 루트보다 나중에 나온다.

in-order에서 v가 몇 번째로 나오는지 저장하는 $\mathrm{Pos}[v]$배열을 관리해주면 쉽게 해결할 수 있다.

```
#include <bits/stdc++.h>
using namespace std;

int N, Pre[1010], In[1010], Pos[1010];

void f(int s, int e, int l, int r){
    if(s > e || l > r) return;
    int root = Pre[s], pos = Pos[root];
```

```
    f(s+1, s+pos-1, l, pos-1);
    f(s+pos-l+1, e, pos+1, r);
    cout << root << " ";
}

void Solve(){
    cin >> N;
    for(int i=1; i<=N; i++) cin >> Pre[i];
    for(int i=1; i<=N; i++) cin >> In[i], Pos[In[i]] = i;
    f(1, N, 1, N);
    cout << "\n";
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int T; cin >> T;
    while(T--) Solve();
}
```

BOJ 10830 행렬 제곱

11819 The Shortest does not Mean the Simplest와 크게 다를 것이 없다.

행렬 곱셈의 항등원은 $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ 형태라는 것을 잊지 말자.

만약 모른다면 A^B를 구하는 대신 $A \cdot A^{B-1}$을 구해도 된다.

```
#include <bits/stdc++.h>
using namespace std;
constexpr int MOD = 1000;

struct Matrix{
    int sz;
    vector<vector<int>> a;
    Matrix(int sz, int init=0) : sz(sz), a(sz, vector<int>(sz)) {
        for(int i=0; i<sz; i++) a[i][i] = init;
    }
};

Matrix Multiply(const Matrix &l, const Matrix &r){
    int n = l.sz;
    Matrix ret(n);
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            for(int k=0; k<n; k++) ret.a[i][j] += l.a[i][k] * r.a[k][j];
            ret.a[i][j] %= MOD;
        }
    }
    return ret;
}

Matrix Pow(Matrix a, long long b){
    Matrix ret(a.sz, 1);
    while(b){
```

```
        if(b & 1) ret = Multiply(ret, a);
        b >>= 1; a = Multiply(a, a);
    }
    return ret;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N; long long B;
    cin >> N >> B;

    Matrix M(N);
    for(int i=0; i<N; i++){
        for(int j=0; j<N; j++) cin >> M.a[i][j];
    }

    Matrix Res = Pow(M, B);
    for(int i=0; i<N; i++){
        for(int j=0; j<N; j++) cout << Res.a[i][j] << " ";
        cout << "\n";
    }
}
```

BOJ 17297 Messi Gimossi

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

int D[41] = {0, 5, 13};

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    for(int i=3; i<=40; i++) D[i] = D[i-1] + D[i-2] + 1;

    int N; cin >> N; N--;
    for(int i=40; i>=2; i--) if(D[i] <= N) N -= D[i] + 1;
    if(N == -1 || N == 5) cout << "Messi Messi Gimossi";
    else cout << "Messi Gimossi"[N];
}
```

BOJ 1517 버블 소트

$i < j \wedge A_i > A_j$인 순서쌍 (i, j)의 개수를 구하는 문제다. 병합 정렬을 응용해서 문제를 해결할 수 있다.

```
#include <bits/stdc++.h>
using namespace std;

int N, A[505050], T[505050];
long long Inv;

void Merge(int s, int m, int e){
    int i = s, j = m+1, idx = s, cnt = 0;
    while(i <= m && j <= e){
        if(A[i] <= A[j]) T[idx++] = A[i++], Inv += cnt;
        else T[idx++] = A[j++], cnt++;
```

```
    }
    while(i <= m) T[idx++] = A[i++], Inv += cnt;
    while(j <= e) T[idx++] = A[j++];
    for(int k=s; k<=e; k++) A[k] = T[k];
}

void MergeSort(int s, int e){
    if(s >= e) return;
    int m = s + e >> 1;
    MergeSort(s, m);
    MergeSort(m+1, e);
    Merge(s, m, e);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N; for(int i=1; i<=N; i++) cin >> A[i];
    MergeSort(1, N);
    cout << Inv;
}
```

BOJ 1725 히스토그램

강의 슬라이드 참고

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

ll N, A[101010];

ll Solve(int s, int e){
    if(s > e) return 0;
    int m = s + e >> 1;
    int l = m, r = m;
    ll now = A[m], h = A[m];
    while(l > s && r < e){
        if(min(h, A[l-1]) > min(h, A[r+1])) h = min(h, A[--l]);
        else h = min(h, A[++r]);
        now = max(now, h * (r-l+1));
    }
    while(l > s) h = min(h, A[--l]), now = max(now, h * (r-l+1));
    while(r < e) h = min(h, A[++r]), now = max(now, h * (r-l+1));
    return max({ now, Solve(s, m-1), Solve(m+1, e) });
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N; for(int i=1; i<=N; i++) cin >> A[i];
    cout << Solve(1, N);
}
```

BOJ 2261 가장 가까운 두 점

강의 슬라이드 참고

```
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;

using ll = long long;
using Point = pair<ll, ll>;
const ll INF = 0x3f3f3f3f3f3f3f3f;

ll Dist(const Point &p1, const Point &p2){
    ll dx = p1.x - p2.x, dy = p1.y - p2.y;
    return dx * dx + dy * dy;
}

int N;
Point A[101010], T[101010];

ll DnC(int s, int e){
    if(s == e) return INF;
    int m = (s + e) / 2;
    ll divLine = A[m].x;
    ll d = min(DnC(s, m), DnC(m+1, e));

    int l = s, r = m+1, idx = s;
    while(l <= m && r <= e){
        if(A[l].y < A[r].y) T[idx++] = A[l++];
        else T[idx++] = A[r++];
    }
    while(l <= m) T[idx++] = A[l++];
    while(r <= e) T[idx++] = A[r++];
    for(int i=s; i<=e; i++) A[i] = T[i];

    vector<Point> now;
    for(int i=s; i<=e; i++){
        ll dx = A[i].x - divLine;
        if(dx * dx < d) now.push_back(A[i]);
    }

    ll ret = d;
    for(int i=1; i<now.size(); i++){
        for(int j=i-1; j>=0; j--){
            ll dy = now[i].y - now[j].y;
            if(dy * dy >= d) break;
            ret = min(ret, Dist(now[i], now[j]));
        }
    }
    return ret;

}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
```

```
    for(int i=1; i<=N; i++) cin >> A[i].x >> A[i].y;
    sort(A+1, A+N+1);
    cout << DnC(1, N);
}
```

BOJ 17613 점프

$J(N)$을 구하는 방법을 먼저 알아보자.
$2^k - 1 \le N$을 만족하는 가장 큰 자연수 k에 대해 $J(N) = k + J(N - 2^k + 1)$이 성립한다. ($1, 2, \cdots, 2^{k-1}$만큼 총 k번 이동)

문제에서는 $J(s), J(s+1), \cdots, J(e)$ 의 최댓값을 요구한다.
k를 고정해서 생각해보자. $2^i - 1 \le x$를 만족하는 가장 큰 자연수 i가 k가 되는 x는 $2^k - 1 \le x < 2^{k+1} - 1$이다. 구간 $[s, e]$를 $k = 1, 2, 3, \cdots$인 구간으로 나누게 되면 모든 쿼리는 최대 $O(\log e - s + 1)$개의 구간으로 분할할 수 있다.

$\log_2 10^9 < 2^{30}$이므로 k는 0부터 29까지만 보면 된다. 중복 계산을 피하기 위해 `std::map`을 이용해 쿼리의 결과를 메모이제이션하면 연산 횟수를 줄일 수 있다.

```
#include <bits/stdc++.h>
using namespace std;
using PII = pair<int, int>;

map<PII, int> D;

int f(int l, int r){
    if(l == 0 && r == 0) return 0;
    if(D.find(PII(l,r)) != D.end()) return D[{l,r}];

    int ret = 0;
    for(int i=0; i<30; i++){
        int s = (1 << i) - 1, e = (1 << (i+1)) - 2;
        s = max(s, l); e = min(r, e);
        if(s <= e) ret = max(ret, i + f(s-(1<<i)+1, e-(1<<i)+1));
    }
    return D[{l,r}] = ret;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int T; cin >> T;
    while(T--){
        int a, b; cin >> a >> b;
        cout << f(a, b) << "\n";
    }
}
```

BOJ 1848 동굴 탐험

1번 정점으로 다시 돌아오기 **직전**에 방문하는 정점 i를 고정해보자. 1번 정점과 i번 정점을 연결하는 간선을 제거하면, (1번 정점에서 i번 정점으로 가는 최단 거리) + (제거한 간선의 가중치)의 최솟값이 문제의 정답이 된다. 이때 시간 복잡도는 $O(NM \log N)$이다.

1번 정점에서 시작해서 **처음으로 방문**하는 정점 $j(\neq i)$도 고정하고, 1번 정점과 j번 정점을 연결하는 간선도 제거해보자. (j번 정점에서 i번 정점으로 가는 최단 거리) + (제거한 두 간선의 가중치 합)의 최솟값이 문제의 정답이 된다. 이때 시간 복잡도는 $O(N^2 M \log N)$이다. 이 풀이를 최적화하면 $O(M \log N \log N)$에 문제를 해결할 수 있다.

$i \neq j$라는 것은 i와 j를 이진법으로 나타내었을 때 한 개 이상의 비트가 다르다는 것을 의미한다.

1번 정점과 연결된 정점들을 $V_1, V_2, \cdots, V_k$라고 하자.

- 만약 i의 x번째 비트가 켜져있다면 V_i를 A그룹, 꺼져있다면 B그룹에 넣고,
- 1번 정점에서 A그룹으로 가는 간선을 모두 제거한 뒤
- A그룹에서 B그룹으로 가는 최단 거리를 구하면 된다.

이 과정을 모든 비트에 대해 전부 시도하면 $O(M \log N)$ 다익스트라를 $O(\log N)$번 수행하게 되므로 $O(M \log^2 N)$에 문제를 해결할 수 있다.

```cpp
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using ll = long long;
using PLL = pair<ll, ll>;
constexpr ll INF = 0x3f3f3f3f3f3f3f3f;

int N, M;
vector<PLL> G[5050];
vector<int> Start;
int W1[5050], W2[5050];
ll D[2][5050];

void Dijkstra(ll dst[5050], priority_queue<PLL, vector<PLL>, greater<>> &pq){
    while(pq.size()){
        auto [cst,v] = pq.top(); pq.pop();
        if(cst > dst[v]) continue;
        for(auto i : G[v]){
            if(dst[i.x] > cst + i.y) pq.emplace(dst[i.x] = cst + i.y, i.x);
        }
    }
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=1; i<=M; i++){
        int s, e, x, y; cin >> s >> e >> x >> y;
        if(e == 1) swap(s, e), swap(x, y);
        if(s == 1) Start.push_back(e), W1[e] = x, W2[e] = y;
        else G[s].emplace_back(e, x), G[e].emplace_back(s, y);
    }

    ll mn = INF;
    priority_queue<PLL, vector<PLL>, greater<>> Q[2];

    for(int bit=0; bit<13; bit++){
        memset(D, 0x3f, sizeof D);
        for(auto i : Start){
            if(i & (1 << bit)) D[0][i] = W1[i], Q[0].emplace(W1[i], i);
            else D[1][i] = W2[i], Q[1].emplace(W1[i], i);
```

```
        }
        Dijkstra(D[0], Q[0]);
        Dijkstra(D[1], Q[1]);
        for(auto i : Start){
            if(i & (1 << bit)) mn = min(mn, D[1][i] + W2[i]);
            else mn = min(mn, D[0][i] + W2[i]);
        }
    }
    cout << mn;
}
```

BOJ 16901 XOR MST

각 가중치는 30비트로 표현할 수 있다. xor로 표현되는 가중치를 최소화하는 것이 목적이기 때문에 Binary Trie를 생각해볼 수 있다.

Boruvka Algorithm과 유사한 방식으로 MST를 구축한다.
먼저, 2^{29}를 나타내는 비트가 켜져 있는 정점 집합과 꺼져 있는 정점 집합을 연결하는 간선은 **한 개**만 있는 것이 최적이다. 비트가 켜져 있는 원소들로 Binary Trie를 구축한 뒤, 꺼져 있는 원소와 xor했을 때 최소가 되는 값을 찾아서 연결하면 된다.

2^{29}를 나타내는 비트의 상태가 서로 다른 집합이 연결되었다. 이제, 두 집합을 따로 처리해줘도 괜찮다.
$2^{28}, 2^{27}, \cdots, 2^0$을 나타내는 비트에 대해 똑같은 과정을 수행하면 된다.

```
#include <bits/stdc++.h>
using namespace std;
constexpr int MX_BIT = 29;

struct Node{
    int l, r;
    Node(){ l = r = -1; }
} T[202020 * MX_BIT];
int nd_cnt;

void Insert(int v, int node=0, int dep=MX_BIT){
    if(dep == -1) return;
    if(v & (1 << dep)){
        if(T[node].r == -1) T[node].r = ++nd_cnt, T[nd_cnt] = Node();
        Insert(v, T[node].r, dep-1);
    }
    else{
        if(T[node].l == -1) T[node].l = ++nd_cnt, T[nd_cnt] = Node();
        Insert(v, T[node].l, dep-1);
    }
}

// return min(v ^ element)
int Query(int v, int node=0, int dep=MX_BIT){
    if(dep == -1) return 0;
    if(v & (1 << dep)){
        if(T[node].r != -1) return Query(v, T[node].r, dep-1);
        else return (1 << dep) + Query(v, T[node].l, dep-1);
    }
    else{
        if(T[node].l != -1) return Query(v, T[node].l, dep-1);
        else return (1 << dep) + Query(v, T[node].r, dep-1);
    }
```

```
}

int N, A[202020];
long long R;

void Solve(int s, int e, int dep=MX_BIT){
    if(dep == -1 || s >= e) return;

    int m = s-1;
    while(m+1 <= e && (A[m+1] & (1 << dep)) == 0) m++;
    Solve(s, m, dep-1); Solve(m+1, e, dep-1);
    if(s > m || m+1 > e) return;

    int now = 0x3fffffff;
    nd_cnt = 0; T[0] = Node();
    for(int i=s; i<=m; i++) Insert(A[i]);
    for(int i=m+1; i<=e; i++) now = min(now, Query(A[i]));
    R += now;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N; for(int i=1; i<=N; i++) cin >> A[i];
    sort(A+1, A+N+1);
    Solve(1, N);
    cout << R;
}
```

BOJ 20503 Haven

BOJ 16901 XOR MST의 풀이를 응용해 MST가 주어졌을 때 X_i를 복원하는 방법을 생각해보자.

$f(T, D)$를 트리 T의 각 정점을 2^{D+1} 미만의 수로 채우는 함수라고 정의하자. T에서 적당한 간선 (u, v)를 선택해서 끊으면 트리가 두 개로 분할된다. 각각 T_1, T_2라고 하면, 아래 과정을 통해 X_i를 복원할 수 있다.

1. T_1에 속한 정점들의 D번째 비트를 0으로 설정
2. T_2에 속한 정점들의 D번째 비트를 1로 설정
3. $f(T_1, D-1), f(T_2, D-1)$ 호출
4. T_1, T_2를 분할하는 간선 (u, v)가 $T1, T2$를 연결하는 최소 가중치 간선이 되도록 $T1, T2$의 각 정점에 적절한 값을 xor

모든 정점의 차수가 3 이하인 트리에는 아래 조건을 만족하는 간선 e가 반드시 존재하기 때문에 최대 28단계 안에 분할 정복이 종료된다.

- 트리 T에서 e를 제거해서 분할된 트리를 $T1, T2$라고 할 때
- $|T_1| \le 2|T_2| + 1, |T_2| \le 2|T_1| + 1$
- 중국에서는 Centroid decomposition on edges라는 이름으로 잘 알려져 있는 것 같다. https://codeforces.com/blog/entry/58372

```
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using PII = pair<int, int>;
constexpr int MX_BIT = 28;
```

```cpp
int N, A[202020], S[202020];
vector<int> G[202020];

int get_sz(int v, int b=-1){
    S[v] = 1;
    for(auto i : G[v]) if(i != b) S[v] += get_sz(i, v);
    return S[v];
}

PII get_edge(int v, int total, int b=-1){
    int now = abs(total - S[v]*2);
    PII ret(b, v);
    for(auto i : G[v]){
        if(i == b) continue;
        PII t = get_edge(i, total, v);
        int nxt = abs(total - S[t.y]*2);
        if(nxt < now) now = nxt, ret = t;
    }
    return ret;
}

void get_vertices(int v, vector<int> &lst, int b=-1){
    lst.push_back(v);
    for(auto i : G[v]) if(i != b) get_vertices(i, lst, v);
}

void Solve(int v, int base=0, int dep=MX_BIT){
    if(get_sz(v) == 1){
        if(A[v] == -1) A[v] = base;
        return;
    }
    auto [s,e] = get_edge(v, S[v]);
    G[s].erase(find(G[s].begin(), G[s].end(), e));
    G[e].erase(find(G[e].begin(), G[e].end(), s));

    vector<int> vertices; get_vertices(e, vertices);
    Solve(s, base, dep-1);
    Solve(e, base | (1 << dep), dep-1);

    int color = A[s] ^ A[e] ^ (1 << dep);
    for(auto i : vertices) A[i] ^= color;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<N; i++){
        int s, e; cin >> s >> e;
        G[s].push_back(e); G[e].push_back(s);
    }
    memset(A, -1, sizeof A);
    Solve(1);
    for(int i=1; i<=N; i++) cout << A[i] << " ";
}
```

BOJ 2042 구간 합 구하기

세그먼트 트리 구현 문제

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

constexpr int SZ = 1 << 20;

int N, M, K;
ll A[SZ], T[SZ << 1];

void Build(){
    for(int i=1; i<=N; i++) T[i|SZ] = A[i];
    for(int i=SZ-1; i>=1; i--) T[i] = T[i << 1] + T[i << 1 | 1];
}

void Update(int x, ll v){
    x |= SZ; T[x] = v;
    while(x >>= 1) T[x] = T[x << 1] + T[x << 1 | 1];
}

ll Query(int l, int r){
    l |= SZ; r |= SZ;
    ll ret = 0;
    while(l <= r){
        if(l & 1) ret += T[l++];
        if(~r & 1) ret += T[r--];
        l >>= 1; r >>= 1;
    }
    return ret;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M >> K;
    for(int i=1; i<=N; i++) cin >> A[i];
    Build();
    for(int i=1; i<=M+K; i++){
        ll op, a, b; cin >> op >> a >> b;
        if(op == 1) Update(a, b);
        else cout << Query(a, b) << "\n";
    }
}
```

BOJ 14438 수열과 쿼리 17

세그먼트 트리 구현 문제

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

constexpr int SZ = 1 << 18;
```

```
int N, Q, A[SZ], T[SZ << 1];

void Build(){
    memset(T, 0x3f, sizeof T);
    for(int i=1; i<=N; i++) T[i|SZ] = A[i];
    for(int i=SZ-1; i>=1; i--) T[i] = min(T[i << 1], T[i << 1 | 1]);
}

void Update(int x, ll v){
    x |= SZ; T[x] = v;
    while(x >>= 1) T[x] = min(T[x << 1], T[x << 1 | 1]);
}

int Query(int l, int r){
    l |= SZ; r |= SZ;
    int ret = 0x3f3f3f3f;
    while(l <= r){
        if(l & 1) ret = min(ret, T[l++]);
        if(~r & 1) ret = min(ret, T[r--]);
        l >>= 1; r >>= 1;
    }
    return ret;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i];
    Build();
    cin >> Q;
    for(int i=1; i<=Q; i++){
        int op, a, b; cin >> op >> a >> b;
        if(op == 1) Update(a, b);
        else cout << Query(a, b) << "\n";
    }
}
```

BOJ 2243 사탕상자

K번째 값을 구하는 세그먼트 트리

```
#include <bits/stdc++.h>
using namespace std;
constexpr int SZ = 1 << 20;

int N, T[SZ << 1];

void Update(int x, int v){
    x |= SZ; T[x] += v;
    while(x >>= 1) T[x] += v;
}
int Kth(int k){
    int x = 1;
    while(x < SZ){
        if(k <= T[x << 1]) x = x << 1;
        else k -= T[x << 1], x = x << 1 | 1;
    }
```

```
    return x ^ SZ;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++){
        int op; cin >> op;
        if(op == 1){
            int a; cin >> a;
            int res = Kth(a);
            cout << res << "\n";
            Update(res, -1);
        }
        else{
            int a, b; cin >> a >> b;
            Update(a, b);
        }
    }
}
```

BOJ 5419 북서풍

서쪽에 있는 점이 앞에 오도록(x좌표 오름차순), x좌표가 같다면 북쪽에 있는 점이 앞에 오도록(y좌표 내림차순) 정렬하자.

자신보다 서쪽에 있는 점은 모두 앞에 있으므로 y좌표만 신경쓰면 된다. 구체적으로, 점들을 순서대로 보면서 y좌표가 자신보다 크거나 같은 점들의 개수를 세어주면 된다.

세그먼트 트리로 쉽게 구할 수 있다.

```
#include <bits/stdc++.h>
#define x first
#define y second
#define all(v) v.begin(), v.end()
#define compress(v) sort(all(v)), v.erase(unique(all(v)), v.end())
#define IDX(v, x) (lower_bound(all(v), x) - v.begin())
using namespace std;
using PII = pair<int, int>;
constexpr int SZ = 1 << 18;

int N, T[SZ << 1];
PII A[SZ];

void Add(int x){
    x |= SZ; T[x]++;
    while(x >>= 1) T[x]++;
}
int Query(int l, int r){
    l |= SZ; r |= SZ; int ret = 0;
    while(l <= r){
        if(l & 1) ret += T[l++];
        if(~r & 1) ret += T[r--];
        l >>= 1; r >>= 1;
    }
    return ret;
}
```

```
void Solve(){
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i].x >> A[i].y;
    sort(A+1, A+N+1, [](const PII &a, const PII &b){
        if(a.x != b.x) return a.x < b.x;
        return a.y > b.y;
    });
    memset(T, 0, sizeof T);

    vector<int> Y;
    for(int i=1; i<=N; i++) Y.push_back(A[i].y);
    compress(Y);
    for(int i=1; i<=N; i++) A[i].y = IDX(Y, A[i].y) + 1;

    long long ans = 0;
    for(int i=1; i<=N; i++){
        ans += Query(A[i].y, Y.size());
        Add(A[i].y);
    }
    cout << ans << "\n";
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int TC; cin >> TC;
    while(TC--) Solve();
}
```

BOJ 2336 굉장한 학생

자신보다 3개의 시험을 모두 잘 본 학생이 없는 학생을 굉장한 학생이라고 할 때, 굉장한 학생의 수를 구하는 문제다.

첫 번째 시험의 등수를 기준으로 학생을 정렬하자. 앞에 있는 학생은 뒤에 있는 학생보다 첫 번째 시험을 잘 봤다는 것을 의미한다. 그러므로 자신보다 앞에 있는 학생 중 두 번째 시험과 세 번째 시험을 모두 잘 본 학생이 없다면 해당 학생은 굉장한 학생이다.

학생들을 순서대로 보면서 구간의 최솟값을 관리하는 세그먼트 트리를 이용해 다음과 같은 연산을 수행하자.

- 구간 [1, 두 번째 시험 등수]의 최솟값이 자신의 세 번째 시험 등수보다 작다면 나보다 모든 시험을 잘 본 학생이 존재한다는 것을 의미한다. 구간의 최솟값이 자신의 세 번째 등수보다 큰지 확인해서 굉장한 학생인지 판별하면 된다.
- 세그먼트 트리의 (두 번째 시험 등수)번째 리프 정점의 값을 세 번째 시험의 등수로 갱신한다.

```
#include <bits/stdc++.h>
using namespace std;
constexpr int SZ = 1 << 19;

struct Point{
    int x, y, z;
    bool operator < (const Point &t) const {
        return tie(x,y,z) < tie(t.x,t.y,t.z);
    }
};
```

```
int N, T[SZ << 1];
Point A[SZ];

void Update(int x, int v){
    x |= SZ; T[x] = v;
    while(x >>= 1) T[x] = min(T[x<<1], T[x<<1|1]);
}

int Query(int l, int r){
    l |= SZ; r |= SZ; int ret = 0x3f3f3f3f;
    while(l <= r){
        if(l & 1) ret = min(ret, T[l++]);
        if(~r & 1) ret = min(ret, T[r--]);
        l >>= 1; r >>= 1;
    }
    return ret;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1,t; i<=N; i++) cin >> t, A[t].x = i;
    for(int i=1,t; i<=N; i++) cin >> t, A[t].y = i;
    for(int i=1,t; i<=N; i++) cin >> t, A[t].z = i;
    sort(A+1, A+N+1);

    int R = 0;
    memset(T, 0x3f, sizeof T);
    for(int i=1; i<=N; i++){
        if(Query(1, A[i].y) > A[i].z) R++;
        Update(A[i].y, A[i].z);
    }
    cout << R;
}
```

BOJ 10167 금광

직사각형의 네 변에 모두 하나 이상의 금광이 걸쳐 있는 경우만 고려해도 항상 정답을 찾을 수 있다. x, y 좌표를 각각 압축하면 x, y 좌표를 1 ~ 2N 사이로 바꿀 수 있다.

직사각형의 왼쪽 변과 오른쪽 변의 x좌표를 고정하자. 총 $O(N^2)$가지이다. 만약 왼쪽 변과 오른쪽 변이 고정되어 있을 때의 최댓값을 $T(N)$ 시간에 구할 수 있다면 문제를 $O(N^2T(N))$시간에 해결할 수 있다.

왼쪽 변과 오른쪽 변이 고정되어 있을 때 직사각형의 최댓값을 구하는 것은, 점들을 y좌표 기준으로 정렬해놓은 리스트에서 가중치의 합이 가장 큰 부분 배열을 구하는 것과 동치이다.

세그먼트 트리를 이용해 최대 부분합을 구하는 방법은 잘 알려져 있으므로 $O(N^2 \log N)$에 문제를 해결할 수 있다.

y좌표가 같은 점은 세그먼트 트리에 동시에 넣어야 한다는 것을 주의하자.

```
#include <bits/stdc++.h>
#define all(v) v.begin(), v.end()
#define compress(v) sort(all(v)), v.erase(unique(all(v)), v.end())
#define IDX(v, x) (lower_bound(all(v), x) - v.begin())
using namespace std;
using ll = long long;
constexpr int SZ = 1 << 12;
```

```cpp
struct Point{
    ll x, y, w;
    bool operator < (const Point &p) const {
        if(x != p.x) return x < p.x;
        return y < p.y;
    }
};

struct Node{
    ll l, r, mx, sum;
    Node() : Node(0) {}
    Node(ll v) : l(v), r(v), mx(v), sum(v) {}
    void set(ll v){ l = r = mx = sum = v; }
    void add(ll v){ l += v; r += v; mx += v; sum += v; }
    Node operator + (const Node &t) const {
        Node ret;
        ret.l = max(l, sum + t.l);
        ret.r = max(r + t.sum, t.r);
        ret.mx = max({ mx, t.mx, r + t.l });
        ret.sum = sum + t.sum;
        return ret;
    }
} T[SZ << 1];

void Init(){
    for(int i=0; i<(SZ<<1); i++) T[i].set(0);
}

void Update(int x, ll v){
    x |= SZ; T[x].add(v);
    while(x >>= 1) T[x] = T[x << 1] + T[x << 1 | 1];
}

int N;
Point A[3030];
vector<int> X, Y;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++){
        cin >> A[i].x >> A[i].y >> A[i].w;
        X.push_back(A[i].x); Y.push_back(A[i].y);
    }
    compress(X); compress(Y);
    for(int i=1; i<=N; i++){
        A[i].x = IDX(X, A[i].x) + 1;
        A[i].y = IDX(Y, A[i].y) + 1;
    }
    sort(A+1, A+N+1);

    ll mx = 0;
    for(int i=1; i<=N; i++){
        if(i > 1 && A[i-1].x == A[i].x) continue;
        Init();
        for(int j=i; j<=N; j++){
            Update(A[j].y, A[j].w);
```

```
            if(j == N || A[j].x != A[j+1].x) mx = max(mx, T[1].mx);
        }
    }
    cout << mx;
}
```

BOJ 16911 그래프와 쿼리

https://koosaga.com/121

```
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using PII = pair<int, int>;

struct UnionFind{
    int P[101010], R[101010];
    UnionFind(){ iota(P, P+101010, 0); fill(R, R+101010, 1); }
    int find(int v){ return v == P[v] ? v : find(P[v]); }
    PII merge(int u, int v){
        u = find(u); v = find(v);
        if(u == v) return {-1, -1};
        if(R[u] > R[v]) swap(u, v);
        P[u] = v;
        if(R[u] == R[v]) R[v]++;
        return {u, v};
    }
    void undo(int u, int v){
        P[u] = u;
        if(R[u]+1 == R[v]) R[v]--;
    }
} uf;

struct Query{
    int u, v, l, r;
    Query() = default;
    Query(int u, int v, int l, int r) : u(u), v(v), l(l), r(r) {}
};

int N, Q, pv = 1;
PII Check[101010];
map<PII, int> mp;
vector<Query> qry;
vector<PII> T[1 << 20];

void Add(int l, int r, PII edge, int node=1, int s=1, int e=pv-1){
    if(r < s || e < l) return;
    if(l <= s && e <= r){
        T[node].push_back(edge); return;
    }
    int m = s + e >> 1;
    Add(l, r, edge, node<<1, s, m);
    Add(l, r, edge, node<<1|1, m+1, e);
}

void DFS(int node=1, int s=1, int e=pv-1){
```

```
    vector<PII> merged;
    for(auto i : T[node]) merged.push_back(uf.merge(i.x, i.y));
    if(s == e){
        if(uf.find(Check[s].x) == uf.find(Check[s].y)) cout << 1 << "\n";
        else cout << 0 << "\n";
    }
    else{
        int m = s + e >> 1;
        DFS(node<<1, s, m);
        DFS(node<<1|1, m+1, e);
    }
    reverse(merged.begin(), merged.end());
    for(auto [u,v] : merged) if(u != -1) uf.undo(u, v);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> Q;
    for(int i=1; i<=Q; i++){
        int op, a, b; cin >> op >> a >> b; if(a > b) swap(a, b);
        if(op == 1) mp[{a, b}] = pv;
        else if(op == 2) qry.emplace_back(a, b, mp[{a,b}], pv),
mp.erase(PII(a,b));
        else if(op == 3) Check[pv++] = {a, b};
    }
    for(auto i : mp) qry.emplace_back(i.x.x, i.x.y, i.y, pv);
    for(auto i : qry) if(i.l < i.r) Add(i.l, i.r-1, {i.u, i.v});
    DFS();
}
```

BOJ 18362 Desert

문제를 간단하게 요약해보면...

> 길이가 N인 수열 A가 있다. 그중 원소 S개는 정확한 값을 알고 있고, 나머지 원소들의 값은 모르는 대신 다음과 같은 정보 M개를 알고 있다.
>
> - $l\ r\ k\ x_1\ x_2\ \cdots\ x_k$: $\min\limits_{i\in\{x_1,x_2,\ldots,x_k\}}(A_i) \geq \max\limits_{i\in[l,r],i\notin\{x_1,x_2,\ldots,x_k\}}(A_i)$
>
> 모든 원소가 10^9 이하의 자연수인 수열 A가 존재한다면 첫 번째 줄에 `TAK`, 두 번째 줄에는 그러한 수열을 아무거나 하나 출력한다. 존재하지 않는다면 첫 번째 줄에 `NIE`를 출력한다.

그래프를 잘 만든 뒤 위상 정렬을 이용해 최장 경로를 구하는 문제다. 풀이가 복잡하니 아래 글을 참고하자.

https://justicehui.github.io/tutorial/2020/09/05/graph-with-segment-tree/

```
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using PII = pair<int, int>;
constexpr int SZ = 1 << 17;
constexpr int MAX_N = 101010, MAX_M = 202020;
constexpr int MAX_V = SZ*2 + MAX_M;

int N, S, M, A[MAX_N], In[MAX_V], D[MAX_V];
```

```cpp
vector<PII> G[MAX_V];

void Init(){
    for(int i=1; i<SZ; i++){
        G[i<<1].emplace_back(i, 0);
        G[i<<1|1].emplace_back(i, 0);
    }
    for(int i=1; i<=S; i++){
        int x, y; cin >> x >> y; A[x] = y;
        G[0].emplace_back(x|SZ, y);
    }
}

vector<int> Range(int l, int r){
    l |= SZ; r |= SZ;
    vector<int> ret;
    while(l <= r){
        if(l & 1) ret.push_back(l++);
        if(~r & 1) ret.push_back(r--);
        l >>= 1; r >>= 1;
    }
    return move(ret);
}

void addEdge(int l, int r, int x){
    for(const auto &i : Range(l, r)) G[i].emplace_back(x, 0);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> S >> M; Init();

    for(int i=1; i<=M; i++){
        int l, r, k; cin >> l >> r >> k;
        for(int j=1; j<=k; j++){
            int t; cin >> t;
            G[SZ*2+i].emplace_back(t|SZ, 1);
            if(l < t) addEdge(l, t-1, SZ*2+i);
            l = t + 1;
        }
        if(l <= r) addEdge(l, r, SZ*2+i);
    }
    for(int i=0; i<MAX_V; i++) for(auto j : G[i]) In[j.x]++;

    queue<int> q;
    for(int i=0; i<MAX_V; i++){
        if(!In[i]) q.push(i);
        if(i) D[i] = 1;
    }
    while(q.size()){
        int v = q.front(); q.pop();
        for(auto i : G[v]){
            D[i.x] = max(D[i.x], D[v] + i.y);
            if(!--In[i.x]) q.push(i.x);
        }
    }

    for(int i=1; i<=N; i++){
```

```
        if(In[i|SZ] || D[i|SZ] > 1e9){ cout << "NIE"; return 0; }
        if(A[i] && A[i] != D[i|SZ]){ cout << "NIE"; return 0; }
    }
    cout << "TAK\n";
    for(int i=1; i<=N; i++) cout << D[i|SZ] << " ";
}
```